Ejemplo de aplicación Java EE 6.
Tienda Web con JPA+EJB+JSF

FJRP – CCIA-2011

Septiembre-2011

Índice

1. Descripción de la aplicación de ejemplo **1**

1.1. Presentación del ejemplo 1

1.2. Pasos previos 2

1.3. Arquitectura de la solución de ejemplo 4

1.4. Simplificaciones 5

2. Capa de lógica de aplicación **5**

2.1. Entidades JPA 6

2.1.1. Entidades 6

2.2. Enterprise Java Beans 6

2.2.1. Paquete *ejb.dao* 7

2.2.2. Paquete *ejb.negocio* 7

3. Capa de presentación Web **8**

3.1. Páginas JSF 8

3.2. Managed Beans 9

4. Capa de presentación SWING **10**

1. Descripción de la aplicación de ejemplo

1.1. Presentación del ejemplo

Como punto de partida para el desarrollo del proyecto Java EE del curso 2011/12 se aporta como ejemplo una pequeña aplicación para la gestión de una tienda web.

Se trata de una aplicación Java EE 6 Cuenta una capa de aplicación (o negocio) implementada con componentes EJB (*Enterprise Java Bean*) que hace uso de JPA (*Java Persistence API*) como mecanismo de mapeo Objeto/Relacional para proveer de una capa de acceso a la base de datos.

Se incluyen 2 capas de presentación diferentes:

- Una capa de presentación Web basada en JSF (*Java Server Faces*) encargada fundamentalmente de la interacción de los compradores, que da soporte a los casos de uso relacionados con la navegación a través del catálogo de la tienda y la confección de pedidos de compra, además del alta de clientes.
- Una capa de presentación consistente en una aplicación de escritorio SWING simplificada encargada de los casos de uso relacionados con el mantenimiento del catálogo de la tienda Web (familias y productos) y la gestión de los pedidos recibidos.

1.2. Pasos previos

1. Instalar el entorno de desarrollo **Netbeans 7.0.1** y el servidor de aplicaciones **GlassFish v3.1.1**

   Es recomendable trabajar con una versión de Netbeans lo más reciente posible (actualmente Netbeans 7.0.1).

   Se puede descargar la última versión de Netbeans en http://www.netbeans.org asegurándose de que incluya el servidor de aplicaciones GlassFish.

   La instalación de Netbeans y Glassfish se hace de forma conjunta quedando Netbeans configurado para usar GlassFish como servidor de aplicaciones por defecto.

   En el laboratorio no se requieren permisos de administrador para instalarlo y debería ser posible mantener a la vez la versión existente junto con la nueva.

   - Para desintalar las versiones antiguas, tanto de Netbeans como de GlassFish, ambas cuentan con un un script bash `unsintall.sh` en sus respectivos directorios de instalación (en `/home/alumno`)
   - También se puede optar por borrar los directorios de instalación sin más, aunque se perderán los accesos directos del menú y habrá que lanzar Netbeans desde línea de comandos (`$ /home/alumno/netbeans-7.0.1/bin/n &`)

2. Instalar la base de datos. Los datos manejados por la aplicación de ejemplo se almacenan en un servidor MySQL.

   Descarga: Tablas+datos (ejemplo-JEE.sql)

   Instalación:

   - Acceder como administrador de MySQL y crear un usuario `scs` con password `scs`

     ```
     $ mysql --user=root --password
     (pedirá el password del administrador, por defecto suele ser el password vacío)
     mysql> GRANT ALL PRIVILEGES ON *.* TO 'scs'@'localhost' IDENTIFIED BY 'scs' WITH GRANT OPTION;
     mysql> GRANT ALL PRIVILEGES ON *.* TO 'scs'@'%' IDENTIFIED BY 'scs' WITH GRANT OPTION;
     mysql> exit
     ```

   - Crear la base de datos `ejemplo_tienda` propiedad del usuario `scs` con la estructura y los datos incluidos en el fichero `ejemplo-JEE.sql`

     ```
     $ mysql --user=scs --password=scs < ejemplo-JEE.sql
     ```

3. Antes de desplegar y ejecutar la aplicación es necesario incluir en el servidor de aplicaciones el conector JDBC de MySQL.

   Se puede usar el fichero `mysql-connector-java-5.1.XX-bin.jar` incluido en Netbeans o descargarlo desde la página de MySQL.

```
$ cp /home/alumno/netbeans-7.0.1/ide/modules/ext/mysql-connector-java-5.1.XX-bin.jar \
     /home/alumno/glassfish-3.1.1/glassfish/domains/domain1/lib/
```

4. Abrir la aplicación en Netbeans La aplicación Java EE de ejemplo está conformada por tres proyectos Netbeans:

   - Un proyecto de tipo *Java EE → EJB Module*
   - Un proyecto de tipo *Java Web → Web Application*
   - Un proyecto SWING (pendiente de incluir)

   Los dos primeros están agrupados en un proyecto *Java EE → Enterprise Application* y se desplarán juntos en el servidor de aplicaciones. Se deberá abrir el proyecto

   Descarga: ejemploTiendaWeb.tar.gz

   ```
   $  tar xzvf EjemploTiendaWeb.tar.gz
   ```

   Al abrir el proyecto *EjemploTiendaWeb* se carga sólo el proyecto principal. Es necesario seleccionar la opción *Open Requierd Projects* en la opción *Open Project* antes de abrirlo o en el menú contextual del proyecto una vez abierto (*[botón derecho] → Open Requierd Projects*), aparecerán 2 subproyectos:

   - *EjemploTiendaWeb-ejb*: módulo EJB con la definición de las entidades JPA y los EJBs de la aplicación
   - *EjemploTiendaWeb-war*: páginas JSF y *Managed Beans* que conforma el interfaz web de la tienda virtual

   En caso de que Netbeans informe del error, se deberá especificar el servidor de aplicaciones a utilizar, en este caso la versión 3.1 de GlassFish.

5. Compilación, despliegue y ejecución.

   - Para compilar la aplicación basta seleccionar la opción *Clean and Build* del proyecto principal (*[botón derecho] → Clean and Build*).
     - En los respectivos directorios `dist` de cada proyecto se generarán los ficheros que empaquetan los componentes de la aplicación y que se deberá desplegar en el servisdor GlassFish
     - Para *EjemploTiendaWeb-ejb* se crea `dist/EjemploTiendaWeb-ejb.jar`
     - Para *EjemploTiendaWeb-war* se crea `dist/EjemploTiendaWeb-war.war`
     - Para *EjemploTiendaWeb* se crea `EjemploTiendaWeb.ear` que agrupa a los dos anteriores
   - Para desplegar la aplicación: sobre el proyecto principal *[botón derecho] → Deploy*
     - En caso de que el servidor GlassFish no estuviera arrancado se iniciará al hacer el primer despliegue
     - Se puede arrancar GlassFish desde linea de comandos

       ```
       /home/alumno/glassfish-3.1.1/bin/asadmin start-domain
       ```

   - Para ejecutar la aplicacion: sobre el proyecto principal *[botón derecho] → Run* (en este caso se lanzará el navegador web por defecto para mostrar la página principal del proyecto *EjemploTiendaWeb-war*)
     - En este caso la aplicación web será accesible en la URL: `http://localhost:8080/EjemploTiendaWeb-war/`

   Desde la pestaña *services* de Netbeans se puede manejar el servidor de aplicaciones GlassFish (*Servers→GlassFish v3 domain*: iniciar, parar, recargar) y ver la lista de aplicaciones deplegadas.

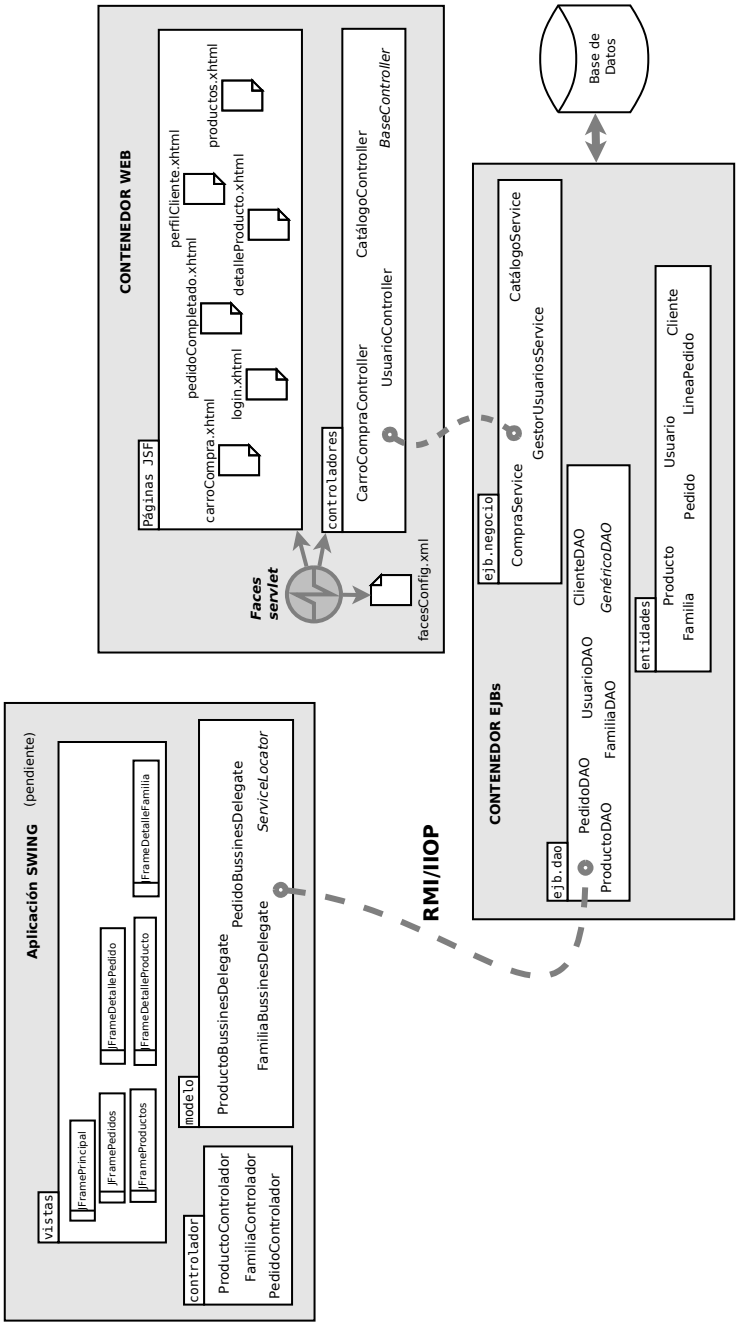
   - En ocasiones durante la depuración será necesario cancelar el despliegue de una aplicación (opción *undeploy* sobre la aplicación concreta)

   También se puede acceder a la consola de GlassFish desde un navegador web con la url: `http://localhost:4848`

   - Desde la opción *Despliegue* dela consola web de GlassFish se pueden desplegar los ficheros `.jar`, `.war` y/o `.ear` de las aplicaciones Java EE.

1.3. Arquitectura de la solución de ejemplo

El ejemplo de aplicación Java EE 6 sigue un esquema de cliente ligero estricto. Tanto la capa de presentación web basada en JSF como la aplicación de escritorio SWING no implementan ningún tipo de lógica de aplicación, limitándose a mostar datos al usuario y a capturar y validar las entradas recibidas. Toda la lógica de la aplicación necesaria para implementar los caso de uso del ejemplo es responsabilidad de los componentes EJB incluidos en el subproyecto *EjemploTienda-ejb*.



En este caso, al tratarse de una aplicación sencilla, esta arquitectura tan estricta puede ser un poco excesiva y en el caso de que sólo se hubieran incluido clientes web se podría hacer conseguido la misma funcionalidad

trabajando exclusivamente en el contenedor de Servlet, con JSF y los *Managed Beans*. En este caso, la razón de incluir una capa de EJBs (y por lo tanto exigir un servidor de aplicaciones JEE completo) es la de permitir el acceso remoto desde la aplicación SWING.

- **Nota:** La versión 3.1 de la especificación EJB (incluida en la futura especificación Java EE 6) contempla la posibilidad de incluir componentes EJBs en la capa web, aunque sólo permite el acceso local desde la JVM donde se ejecuta el servidor de aplicaciones, no el acceso remoto sobre RMI/IIOP desde otras JVMs.

1.4. Simplificaciones

- En la capa de negocio se han definido las Entidades JPA y a partir de ellas se generaron las tablas MySQL (usando los tipos y tamaños por defecto de cada campo)
  - Ver el fichero de configuración `persistence.xml` en el subproyecto *EjemploTiendaWeb-ejb*
  - En un entorno real lo más habitial será el caso contrario, crear las entidades JPA mapeando tablas relacionales ya existentes.
- En la capa de negocio se ha utilizado un EJB con estado (`@Stateful`) para implementar el carro de la compra de cada cliente.
  - En este caso se ha hecho así por motivos didacticos, para mostrar un ejemplo de uso de estos EJBs.
  - Lo más sencillo hubiera sido gestionar el carro de la compra dentro de los *ManagedBeans* de sesión de la capa web, dejando que todas las operaciones realizadas por los EJB fueran sin estado.
- En la capa web se usan los componentes de validación de entrada estándar incluidos en JSF 2.0 (rangos, fechas, etc)
- Las funcionalidades de la capa web se han limitado al máximo para evitar complicar en exceso el ejemplo, por lo que en algunos aspectos (como la gestión de los elementos del carro de la compra) el interfaz de usuario no es todo lo cómodo y funcional que debiera.
- En la capa web todas las páginas siguen la misma plantilla, por lo que las acciones comunes y sus reglas de navegación se repiten en casi todas las páginas.
- En la base de datos y en la capa web (también en la capa EJB) los passwords de los usuarios se almacenan y manejan en claro en forma de strings.
  - Por razones de seguridad lo habitual es almacenar un hash (MD5, SHA-1) del password y no el password en si.
  - Ver la librería Java Simplified Encryptionhttp://www.jasypt.org/ el tutorial How to encrypt user passwords
- En la presentación de los datos en la aplicación web se ha omitido, por simplicidad, la paginación de los resultados. Lo deseable sería contar con un esquema de paginación que evite el tráfico de grandes volúmenes de datos y facilite la navegación.
- Se ha usado una hoja de estilo CSS genérica aplicando modificadores de estilo en componentes puntuales. El esquema ideal hubiera sido delegar completamente la maquetación a las hojas de estilos CSS.

2. Capa de lógica de aplicación

El proyecto *EjemploTiendaWeb-ejb* contiene la definición de las Entidades JPA responsables del mapeo Objeto/Relacional de la base de datos y los EJBs responsable de implementar la lógica de los casos de uso de la aplicación.

2.1. Entidades JPA

El paquete `entidades` contiene la defínción de las clases Entidad del ejemplo, junto con las enumeraciones *TipoUsuario* y *EstadoPedido*.

Todas las entidades (≡ tuplas) están identificadas por atributos numéricos gestionados por MySQL (campos autoincrementales), no se utilizan atributos de las entidades como identificadores (NIF, etc).

2.1.1. Entidades

Usuario. Almacena los datos de los usaurios del sistema (login, password, tipo, fecha de alta, fecha de último acceso). Se usa principalmente para el control de accesos.

Cliente. Almacena los datos personales de los usuarios de tipo cliente.

- Tiene una relación 1:1(*@OneToOne*) con *Usuario*

Producto. Almacena los datos de los productos que conforman el catálogo de la tienda.

- Tiene una relación N:1 (*@ManyToOne*) con *Familia*

Familia. Almacena las descripciones de las familias de productos consideradas en el catálogo de la tienda.

Pedido. Almacena la información de los pedidos (fecha, cliente, estado)

- Tiene una relación N:1 (*@ManyToOne*) con *Cliente*
- Tiene una relación 1:N (*@OneToMany*) con *LineaPedido*
  El acceso a las LineasPedido desde Pedido es de tipo (*fetch=FetchType.LAZY*), lo que significa que las lineas de pedido no se carga desde la base de datos hasta que se pretenda acceder a ellas.
  - Se hace así para evitar un número de consultas y un tráfico excesivo cuando se hacen consultas donde se devuelva multitud de pedidos.
  - Las búsquedas de pedidos devolverán sólo la "cabecera" del pedido, no las líneas de detalle.
  - Para "cargar" las lineas de un pedido es necesario acceder a la lista de LineaPedido (método *getLineas()*) (así se hace cuando se pide un Pedido con un ID concreto en el EJB *PedidosDAO*)

  Todas operaciones (borrado, creación, actualización, ...) sobre el lado 1 (Pedido) se propagan en cascada al lado N (LineaPedido) *(cascade=CascadeType.ALL)*

LineaPedido. Almacena la información de una línea de un pedido.

- Tiene una relación N:1 (*@ManyToOne*) con el *Pedido* del que forma parte
- Tiene una relación N:1 (*@ManyToOne*) con *Producto*

2.2. Enterprise Java Beans

Los EJB que conforman la capa modelo y son responsables de la lógica de la aplicación se reparten en dos paquetes (`ejb.dao`, `ejb.negocio`). En ambos caso se ofrece tanto un interfaz local (usado por la aplicación Web JSF) como un interfaz remoto (usado por la aplicación de escritorio SWING).

- De modo general, la capa de lógica de aplicación sigue la arquitectura descrita en el artículo Lean SOA with Java EE 6
- Los EJBs de ambos paquetes implementan de forma aproximada el patrón Session Facade, ofreciendo a los clientes locales o remotos un interfaz de acceso a la lógica de la aplicación.

- La distinción entre ambos paquetes se debe a su orientación. El paquete `ejb.dao` contiene EJB orientados principalmente a dar soporte a los casos de uso típicos en las tareas de mantenimiento de una Base de Datos (operaciones CRUD [*create, read, update, delete*]: acceso, altas, bajas y modificiaciones). El paquete `ejb.negocio` ofrece soporte a casos de uso un poco más específicos de de más alto nivel que hacen uso los EJBs del anterior paquete.
- En ambos casos se ha seguido la misma convención de nombrado:
  - `XxxxDAO.java` ó `XxxxService.java`: Clase de implementación del EJB (*@Stateless* o *@Stateful*)
  - `XxxxDAOLocal.java` ó `XxxxServiceLocal.java`: Interfaz local del EJB (*@Local*)
  - `XxxxDAORemote.java` ó `XxxxServiceRemote.java`: Interfaz remoto del EJB (*@Remote*)

  Por simplicidad ambas interfaces (local y remota) coinciden, aunque no suele ser lo habitual.
- Salvo *CarroCompraService* todos son EJB sin estado (*@Stateless*).

Hay un tercer paquete (`ejb.excepciones`) con la definición de las excepciones generadas por la capa modelo: `ExcepcionEntidad` (error en el acceso a una entidad de la B.D., uso de un ID que no existe, inserción de un ID repetido, etc), `ExcepcionExistencias` (intento de servir un producto del que no hay stock disponible)

2.2.1. Paquete *ejb.dao*

El paquete `ejb.dao` provee de EJBs para implementar las operaciones básicas sobre las entidades que componenen la aplicación (con la excepción de la entidad *LineaPedido* que es gestionada por el EJB *PedidoDAO*).

La funcionalidad ofrecida por este conjunto de EJB se coresponde de un modo genérico con el patrón DAO (Data Access Object), que oculta las tecnologías y el modo de acceso a los datos, delegando, en este caso, las operaciones concretas en el *EntityManager* de JPA.

En esta caso se ha definido un DAO genérico (*GenericoDAO* [implementación] y *GenericoDAOInterface* [inetrface]) con las 4 operaciones básicas (crear, buscar por ID, actualizar y borrar). De esta clase genérica (junto con el respectico interfaz genñerico) heredan los demás EJBs (y sus interfaces local y remoto), añadiendo nuevas operaciones, usualmente operaciones de búsqueda específicas.

GenericoDAO. Operaciones básicas sobre las entidades (independientes de la Entidad concreta)

UsuarioDAO. Operaciones sobre Usuarios. Incluye operaciones de autenticación y comprobación de privilegios.

ClienteDAO. Operaciones sobre Clientes. Añade diversos tipos y criterios de búsqueda.

ProductoDAO. Operaciones sobre Productos. Añade diversos tipos y criterios de búsqueda, incluida la búsqueda por familia.

FamiliaDAO. Operaciones sobre Familias. Añade diversos tipos y criterios de búsqueda.

PedidoDAO. Operaciones sobre Pedidos (e implicitamente sobre LineaPedido). Añade diversos tipos y criterios de búsqueda y la operación *anularPedido()*, que marca un pedido como anulado y reincorpora las candidades pedidas a las existencias de los productos devueltos.

2.2.2. Paquete *ejb.negocio*

En este paquete se incluyen EJBs que implementan caso de uso específicos de la aplicación. En general proveen de operaciones de mayor complejidad que las del paquete `ejb.dao`, responsabilizándose de coordinar las invocaciones de otros EJBs encargados del manejo de datos. En este caso se implementa un patrón *Service Facade* puro.

CatalogoService. EJB sin estado responsable de la gestión del catálogo de productos, realizando búsquedas de productos y familias bajo diversos criterios. Delega sus tareas en los EJBs *ProductoDAO* y *FamiliaDAO*

CompraService. EJB con estado responsable de la gestión del carro de la compra de un cliente.

- Mantiene la lista de *ProductoCompra* de cada cliente donde se asocia el Producto y la cantidad comprada
- Verifica la disponibilidad de los productos pedidos
- Genera el Pedido (junto con las LineaPedido correspondientes) una vez que se confirma la compra.

Nota: No es estrictamente necesario utilizar un EJB con estado en este caso (además de ser poco escalables), se incluye para mostar su uso.

GestorUsuariosService. EJB sin estado resposable de la autenticación y de la gestión de usuarios y clientes (creación de nuevos Clientes, junto con le alta de su respectivo Usuario, y modificación de los datos del cliente)

3. Capa de presentación Web

La capa de presentación Web se ha implementado utilizando el framework JSF(*Java Server Faces* 2.0). Se ha empleado *Facelets* como tecnología para la definición de las vistas en lugar de páginas JSP(*Java Server Pages*), en primer lugar por ser la tecnología por defecto para JSF 2.0 y por la facilidaes que ofrece para definir y manejar plantillas.

Las responsibilidades de la capa web basada en JSF se distribuyen entre tres componentes:

- Páginas JSF: ficheros XHTML (en el caso de emplear Facelets) donde se define la disposición y propiedades de los componentes JSF de la presentación web.
- *Managed Beans*: clases Java que proveen los datos a presentar en las páginas JSF y los métodos invocados por las acciones desencadenas por los eventos de la página JSF.
- Fichero `faces-config.xml`: define los *Managed Beans* que conforman la aplicación JSF y su alcance (sesión, petición, aplicación).

  Define las reglas de navegación que en conjunción con los valores de retornod de los métodos de acción de estos *Managed Beans* determinan el flujo entre las páginas JSF.

3.1. Páginas JSF

Todas las páginas JSF que conforman la aplicación comparten la misma plantilla (definida en el fichero `plantillas/plantillaGeneral.xhtml` conforme a la sintaxis de los *Facelets*).

- Todas las páginas tienen 4 secciones: encabezado, columna izquierda, contenido y pie de página
- El encabezado y la columna izquierda es común a todas las páginas
  - El encabezado se define en `plantillas/vistaCabecera.xhtml`. Contiene un mensaje de presentación y bienvenida y enlaces para desencadenar las acciones básicas del ususario (login, logout, registro, ver carrito, ver/editar perfil)
  - La comuna izquierda se define en `plantillas/vistaIzquierda.xhtml`. Contiene cajas de búsqueda y una lista de familias con enlaces para que el usuario navege a través del catálogo de productos.
- El pié de página es fijo, contiene únicamente un mensaje.

Cada una de las páginas JSF que componen la aplicación define su propia zona de contenido.

- `productos.xhtml`: presenta una tabla con la lista de productos
- `detalleProducto.xhtml`: presenta los detalles del producto seleccionado
- `carroCompra.xhtml`: presenta la lista de productos incluidos en la compra de un usuario y permite modificarla y/o confirmarla para generar el correspondiente pedido
- `perfilCliente.xhtml`: presenta los datos de un cliente, bien para darlo de alta (registro) o para que él modifique su perfil
- `login.xhtml`: página de login

3.2. Managed Beans

En el ejemplo se definen tres *Managed Beans* dentro del paquete `controladores`. Todos ellos tienen alcance de sesión (*@SessionScoped*), por lo que sus atributos estarán disponibles mientras dure la sesión del usuario (o hasta que esta caduque). El seguimiento de la sesión es responsabilidad del servlet JSF, el programador simplemente debede declarar el tipo de alcance.

Los *Managed Beans* de la aplicación de ejemplo delegan toda su funcionalidad en los EJBs de la capa de aplicación (subproyecto *EjemploTiendaWeb-ejb*) por lo que sus únicas funciones estan relacionadas con la gestión de las interacciones derivadas de la acción del usuario:

- Hacer disponibles los datos a mostar, que a su vez se obtendrán de los EJBs
- Mantener los datos introducidos por el usuario durante su interacción con la aplicación.
- Ofrecer los métodos encargados de gestionar los eventos generados por los compoenentes JSF de la aplicación.
  - Delegan en los EJBs las operaciones de acceso a datos y las responsables de implementar los casos de uso de la aplicación.
  - Determinan el flujo entre páginas JSF mediante los valores de retorno (String) de los manejadore de las acciones (enlaces y botones) desencadenadas por el usaurio.

Todos los *Managed Beans* heredan de la clase *BaseController* que ofrece una serie de funcionalidades básicas comunes a todos los controladores:

- Acceso a los parámetros de las peticiones HTTP
- Inserción de mensajes de error en el árbol de componentes de la página JSF (mediante el *FacesContext*)

Managed Beans de la capa web del ejemplo

CatalogoController. Gestiona las interacciones relacionadas con el catálodo de productos de la tienda (búsqueda de productos, navegación por familias, presentación de detalles del producto)

- Mantiene un lista de las Familias disponibles y la lista de Productos que encajan con los criterios de búsqueda actuales
- Mantiene una referencia al Producto seleccionado actualmente
- Delega sus operaciones en el EJB *ejb.negocio.CatalogoService*

CarroCompraController. Gestiona las interacciones relacionadas con el mantenimiento del carro de la compra del cliente.

- Delega sus operaciones en el EJB *ejb.negocio.CompraService*
- Gestiona la adición y eliminación de productos al carro de la compra del usuario y la generación del pedido definitivo.

UsuarioController. Gestiona las interacciones relacionadas con la autenticación de usuarios y la gestión del perfil del cliente.

- Delega sus operaciones en el EJB *ejb.modelo.GestorUsuariosService*
- Mantiene los datos relacionados con la autenticación del usuario (login, password, objeto Usuario)
- Mantiene los datos del cliente actual una vez que este haya hecho login satisfactoriamente o que se haya registrado como nuevo usuario.

En los métodos de los *Managed Benas* vinculados a acciones de las páginas JSF (atributo *action* en enlaces [`h:commandLink`] y botones [`h:commandButton`]) se ha seguido la convención de que sus nombres comiencen por el prefijo `do`. Por ejemplo:

- `usuarioController.doCrearUsuario()`
- `usuarioController.doLogin()`
- `catalogoController.doBusquedaDescripcion()`)
- etc,...

4. Capa de presentación SWING

(pendiente de completar)